Analyse d'algorithmes - Travail Pratique 1

Organisation

- Ce TP est à réaliser seul.
- Une séance de 3h30 y est dédiée.
- Le TP est à rendre via la plateforme TPLab au plus tard ???.

Ce TP est composé de trois problèmes qui sont indépendants entre eux. Il est toutefois conseillé de les faire dans l'ordre donné.

Livrables

Rendez une archive (au format tar, tgz ou zip uniquement). Cette archive doit contenir 3 dossiers, i.e., un pour chaque problème (ou moins si un problème n'a pas été abordé). Dans chacun des ses dossiers, on trouve :

- Dossier src: dossier contenant les sources de vos programmes,
- Fichier Documentation.pdf: un compte-rendu du problème où seront présentes les différentes réponses aux questions,
- Fichier Etudiant : document texte contenant le nom et l'adresse courriel de l'étudiant concerné.
- Fichier README : document texte expliquant comment compiler et lancer vos programmes.

Remarques

- Pour information, pour le premier problème des bouts de code sont déjà implémentés en python/c++/c#/Java, pour le second problème un bout de code est donné en Python. Cependant, vous pouvez utiliser le langage de programmation de votre choix (et même changer de langages entre différents problèmes). Cela dit, l'enseignant se réserve un droit de veto si le langage n'est pas c, c++, python, java,javascript. Demandez-lui si le langage choisi est adéquat avant de commencer à coder!
- Des pénalités seront appliquées à tout travail dont la qualité n'est pas convenable dans le cadre d'une formation de niveau Master. Voici quelques exemples de pénalités pouvant être appliquées :
 - Code qui ne compile pas (pénalité sévère !!!).
 - Fichier manquant (README, ...).
 - Noms de fichiers inappropriés.

Problème 1 : Se familiariser avec les outils

Fichier graphChronoGenerator.py

Dans ce TP, vous serez amener à utiliser le programme 'graphChronoGenerator.py'. Il vous permet d'exécuter votre algorithme sur des entrées de plus en plus grandes, de tracer la courbe des temps d'exécution et de trouver la courbe 'mathématique' la plus proche de ces points. Il se lance directement sous la commande 'python graphChronoGenerator.py'. Vous ne devriez rien avoir à modifier dans ce programme.

Toutefois vous serez amener à modifier les paramètres dans le fichier 'test-Data.json'.

Énoncé

Construire cinq programmes et tester si leur complexité est bien celle attendue. Pour cela, on pourra s'aider des fichiers donnés sur le wiki:

- 'python3 genererNombres.py 20' devrait créer un fichier (par défaut 'integersList_20', mais peut être modifié par le paramètre '-tempFile') contenant une liste de 20 entiers
- En fonction de votre langage, './votreAlgo integersList_20', 'java votre-Algo integersList_20', 'python votreAlgo.py integersList_20', etc... devrait faire tourner votre algorithme sur cette liste d'entiers
- Après modification des entrées 'executable' et 'generator' du fichier'testData.json' 'python graphChronoGenerator.py' devrait faire tourner votre algorithme sur une série de listes d'entiers de tailles croissantes
- Vous pouvez modifier les paramètres de la commande précédente dans le fichier 'testData.json' de ce dossier (pour plus d'informations, voir à la fin de ce TP).
- Si tout se passe bien, vous ne devriez avoir à modifier que les fichiers 'votreAlgo.*' et 'testData.json'.

Les 5 programmes à implémenter doivent vérifier :

- a) Le premier programme doit tourner en temps linéaire O(n).
- b) Le deuxième programme doit tourner en temps quadratique $O(n^2)$.
- c) Le troisième programme doit tourner en temps cubique $O(n^3)$.
- d) Le quatrième programme doit tourner en temps pseudolinéaire $O(n \log n)$.
- e) Le cinquième programme doit tourner en temps exponentiel.

Problème 2 : Distance Euclidienne minimale dans le plan

Le but de ce problème est d'implémenter et de tester la complexité d'un algorithme vu durant le cours. Donné un fichier contenant une liste de points (coordonnée x, coordonnée y), on veut trouver la distance minimale parmi toutes les paires de points. Le fichier sera formaté de la façon suivante : un point par ligne (à partir de la deuxième ligne); et sur chaque ligne : abscisse ESPACE ordonnée. Il est facile de créer des exemples grâce au fichier 'genererPoints.py'.

- a) Coder une fonction 'distance_min_naive' qui parcourt tous les couples de points et retourne la distance minimale.
- b) Donner la complexité de cet algorithme en fonction de la taille de l'entrée et montrer que cela correspond bien à la vitesse d'exécution du programme (on pourra utiliser graphChronoGenerator.py).
- c) Coder l'algorithme (utilisant le paradigme diviser-pour-régner) vu en cours pour résoudre ce problème (peut être retrouvé sur le wiki).
- d) Tester la vitesse d'exécution de ce dernier algorithme et comparer à la complexité attendue.
- e) Estimer le temps de calcul (en 'prolongeant les courbes') pour les deux algorithmes sur une entrée de taille 10^8 .

Problème 3 : Programmation dynamique

On considère le jeu suivant : un plateau de jeu est formé de n+1 cases consécutives, indicées de 0 à n. On inscrit 0 dans les cases 0 et n, puis on inscrit des nombres entiers positifs choisis aléatoirement dans toutes les autres cases. Au début de la partie, un joueur place son jeton dans la case 0. Le but est d'atteindre la case n en effectuant des pas de 1, 3 ou 5 cases, de manière à minimiser la somme des nombres inscrits dans les cases visitées. Par exemple, pour n=7 on pourrait avoir le plateau suivant :

À partir de la case 0, un joueur peut : atteindre la case 3 en effectuant un pas de 3, puis atteindre la case 6 en effectuant un autre pas de 3 et finalement

atteindre la case 7 en effectuant un pas de 1. Ce joueur aura un score de 6 car il a visité les cases 0, 3, 6 et 7 pour un total de 6.

Mais dans ce cas-ci, une solution optimale aurait consisté à visiter les cases 0, 1, 6 et 7 pour un total de 5.

0	1	2	3	4	5	6	7
0	2	10	3	4	12	3	0
$\overline{}$	<i></i> <			7 🗸			

La fonction sommeMin(t,n) ci-dessous calcule la somme minimale qu'un joueur peut réaliser. En entrée, le tableau t représente un plateau de jeu et n est l'indice de sa dernière case (t contient n+1 cases, de 0 à n). La valeur t[i] est le nombre inscrit dans la i-ième case du plateau de jeu.

(Note: sommeMinRec(t,i) calcule la somme minimale pour atteindre la case i. En particulier, sommeMinRec(t,0) retourne toujours 0 et sommeMinRec(t,n) est la valeur recherchée.)

- a) Coder l'algorithme ci-dessus (on pourra s'aider des fichiers des deux précédents problèmes). L'algorithme devrait prendre en argument un fichier contenant n+1 lignes où la ligne i contient le contenu de la case i (des exemples de tels fichiers sont donnés sur le wiki, et l'on peut en générer grace au script 'genererPartie.py').
- b) Quelle est la taille de l'entrée? Déterminez si la complexité temporelle de la fonction sommeMin est polynomiale en fonction de la taille de l'entrée ou non.

- c) Tester si la vitesse de l'algorithme correspond à la complexité calculée à la question précédente.
- d) Expliquer intuitivement pourquoi l'algorithme est si long
- e) Réécrivez la fonction sommeMin(t,n) en utilisant la programmation dynamique pour améliorer son efficacité.
- f) Calculer la nouvelle complexité temporelle en notation grand-O.
- g) Tester de nouveau si la vitesse d'exécution correspond à la complexité attendue.

Aide à l'utilisation du fichier 'testData.json'

Des précisions sur les items du fichier 'parametresGraphChronoGenerator.json' (ne pas hésiter à demander au professeur des précisions si besoin)

- generator : "cmd_g"

 Le programme lancera la commande 'cmd_g n' où n est un entier (taille de l'entrée du test). Cela créera un fichier de test du nom de 'temp_file'.
- executable : "cmd_e"

 Le programme lancera la commande 'cmd_e temp_file' où 'temp_file' a été crée à la commande précédente.
- temp_file : "calculAux.tmp"

 Nom du fichier 'temp_file'. Attention si vous avez déjà un fichier de ce nom dans votre dossier, l'appel à graphChronoGenerator.py réécrira par dessus.
- valeurInitiale : 10
 Valeur initiale pour n
- borneSup : 1000000 Valeur maximale pour n
- increment : "floor(3*x/2)+1" Incrément de n
- attenteMax : 5
 Si ce temps en secondes est dépassé, on arrête le programme au prochain calcul.

- logScale : false Si c'est à vrai, le graphique aura une échelle logarithmique
- courbesSup : ["log(x)", "x", "x*log(x)", "x**2", "x**3", "2**x"]

 Le programme essayera de trouver parmi une de ces fonctions mathématiques celle qui s'approche le mieux de la courbe des temps d'exécution.
- outputImageNamePrefix : "graph", Nom du fichier pour le graphique.